Chapter 6

Lexicon

The role of the lexicon in the Minimalist Program is to serve as input for the structure-building operation Merge. The structure-building operations Merge and Move take two trees α and β to construct a tree γ with α and β as its left and right subtrees. The input trees for the operation Merge can either be lexical items or trees already built by Merge and Move (see also Section 1.6).

The formalization, which is representational instead of derivational, enables to judge if trees are correct according to the Minimalist Program. The structure-building operations Merge and Move do not play a role. The formalization defines the requirements an LF-tree must meet, not the way it is built up. The lexicon is linked with the rest of the formalization via $\overline{\text{X}}$-Theory (see Chapter 7). Within the $\overline{\text{X}}$-module of the formalization, it is defined that heads must be associated with a feature structure that is a member of the lexicon. Note that also in the derivational version the lexicon and $\overline{\text{X}}$-Theory are closely related, since the operations Move and Merge build structures that are permitted by $\overline{\text{X}}$-Theory.

In Section 6.1 I will outline the ideas on the lexicon on which the formalization is based. The formalization itself is discussed in Section 6.2.

6.1 Introduction

Phonological, semantic and formal features In the Minimalist Program, the lexicon is considered to be a set of lexical elements represented by bundles of features. In Chomsky's 1995 framework [Cho95] features are divided into three categories: *phonological*, *semantic* and *formal* (or syntactic) features. Phonological features are interpreted at PF, semantic features are interpreted at LF, and formal features trigger the movements that take

place during overt and covert syntax.

As we saw in Chapter 5, what are called bundles of features in the minimalist framework are treated as feature structures in the formalization. Hence, lexical items are represented as feature structures in the formalization. And furthermore, in our framework formal features do not trigger movements. Since the formalization is non-derivational, movements are modelled by declaratively defined 'licensing chains'. A chain may have a copy in a given position within a functional projection if the formal features of the copy can be checked against the formal features of the relevant functional head.

Formal features are split up in two categories according to Chomsky [Cho95]: optional and intrinsic. Optional features are called variable features by Zwart [Zwa97] for reasons that will be explained later in this section.

According to Chomsky [Cho95, Page 236], formal features are intrinsic if they are idiosyncratic or if they can be predicted from other properties (semantic features) of the lexical item.

An example of an idiosyncratic feature is grammatical gender for the Dutch word *klok* (clock). The fact that *klok* is feminine is not predictable from any semantic feature of the word and therefore this [gender] feature must be listed explicitly in the lexicon. Note that Dutch does not have a feminine/masculine distinction with respect to the selection of determiners. Both feminine and masculine nouns select the definite article *de* while neuter nouns select the definite article *het*.

An example of a predictable feature is [person]. The fact that *klok* is third person is predictable from the semantic feature [artifact]. Therefore [person] is also considered to be intrinsic to *klok*, like [gender]. Chomsky assumes that features such as [person], that can be determined by semantic features, are not listed explicitly in the lexical entry. These features are added to the lexical item when it enters the derivation.

Zwart's view differs from Chomsky's view at this point. Zwart assumes that both types of intrinsic features (idiosyncratic features and features derivable from semantic features) should be listed explicitly in the lexical item.

Furthermore Zwart does not consider the [category] feature to be a formal feature. This feature is grouped together with semantic features in a category of features that is referred to as *lexical-categorial* features (henceforth LC-features). The reason for calling the [category] feature an LC-feature is that it is derivable from semantic features. For instance, we do not need to stipulate a formal feature [category noun] since we know that all lexical items with the semantic feature [artifact] automatically are nouns [Zwa97, Page 169].

Optional features are all formal features that are not intrinsic. Zwart [Zwa97, Page 170-171] claims that 'optional' is the wrong name for this type of feature. It suggests that the presence of the feature is optional whereas its presence is no more optional than the presence of intrinsic features. The real difference between intrinsic features and what Chomsky calls optional features is that optional features have a variable value whereas the values of intrinsic features are fixed.[1] Therefore Zwart prefers to refer to optional features as variable features. Examples of variable (or optional) features for a noun are [case] and [number]. For instance, the [case] of the Dutch word *stoel* (chair) can either be nominative or accusative.

Traditionally the lexicon is the location for everything in a language that is unpredictable.[2] In the same spirit, Chomsky claims that variable (optional) features are first added to the lexical entry as it enters the derivation, since those features are predictable from other properties of the lexical entry (see earlier in this section). Zwart reformulates this idea in the following way: the value of a variable feature is fixed as it enters the derivation, i.e. at this point a choice from the possible values is made. The difference between the two approaches is that in Zwart's point of view the lexicon contains both intrinsic and variable formal features, whereas it only contains intrinsic formal features in Chomsky's point of view.

Postlexicalism There is also a difference between Chomsky's approach and Zwart's approach in the presence of phonological features in the lexicon. Chomsky assumes that lexical entries contain phonological, semantic and intrinsic formal features. Zwart assumes that lexical entries only contain semantic and formal features. This approach is called *postlexicalism*. In postlexicalism (introduced as *Distributed Morphology* by Halle and Marantz [HM93]) phonological features, which determine how a lexical item is pronounced, are added after the syntactic derivation has been completed (cf. also [Bea66, Aro76, Bea91, And92, Aro92]).

Zwart [Zwa97, Page 162-165] advances several arguments (of which, for the sake of simplicity, only one will be described here) to adopt this approach. One of the arguments is connected with Economy of Representation. Because phonological features are not relevant to syntax it is more economical if phonological features are absent during the syntactic derivation. In the postlexicalist approach as opposed to the lexicalist view that is adopted by Chomsky [Cho93], phonological features are added after Spell-Out.[3] Therefore postlexicalism is more economical than lexicalism, in which

[1] There are exceptions to this rule. For example, the Dutch word *fiets* (bicycle) can either be masculine or feminine, although [gender] is an intrinsic feature.

[2] Cf. [Blo33, Page 274] and [Aro92].

[3] In the lexicalist approach, lexical items enter the derivation in fully inflected form. The syntax cannot manipulate inflectional affixes of words.

phonological features are present throughout the syntactic derivation (cf. [Zwa97, Page 160-165]).

Zwart assumes that in the morphological component after Spell-Out, the features of the morphosyntactic object determine the phonological features of the object, i.e. the actual word as it appears at PF. It is claimed that the morphological component can be considered as a kind of lexical insertion that takes place after the derivation [Zwa97, Page 161]. After syntax, a choice (based on morphosyntactic objects) is made from a postsyntactic lexicon. On the basis of semantic features (such as [artifact]) and formal features (such as [person] and [number]) the phonological features of the morphosyntactic object are selected.

Possibly there will be more than one match between the morphosyntactic object and the postsyntactic lexicon. In such a case the most specific entry will be selected. Zwart [Zwa97, Page 164] clarifies this idea with the words in Example 6.1 and 6.2. Example 6.1 gives a possible lexical representation for singular forms of the Dutch verb *kussen* (to kiss). Example 6.2 gives the representation of the forms in Example 6.1 as they appear in the underspecified postsyntactic lexicon.

Example 6.1

[number singular], [person 1]	kus
[number singular], [person 2]	kus**t**
[number singular], [person 3]	kus**t**

Example 6.2

[number singular], [person 1]	kus
[number singular]	kus**t**

The lexical entries in the postsyntactic lexicon are underspecified because this is the most economical way of representing morphological paradigms (cf. [Kip73] and [HM93]). If the relevant morphosyntactic object is first person, both *kus* en *kust* in Example 6.2 will match. As noted above, the closest match is the right match. Therefore *kus* is selected from the postsyntactic lexicon.

In the formalization two different lexicons are defined: one on which the input of the LF-tree are based (henceforth: the prelexicon) and one which is consulted at PF (henceforth: the postlexicon).

Disjunctive feature values In the lexicon we apply feature values that are disjunctions (of either atomic values or feature values that are feature structures).[4] This implies that what looks like a feature structure in the

[4] Cf. [Shi86, Page 14].

lexicon might actually be a description of a set of feature structures. I will clarify this with the help of the simplified feature structure representing a lexical item for the Dutch noun *boek* (book) in Example 6.3.

Example 6.3

$$\begin{bmatrix} \text{WORD} & \textit{boek} \\ \text{CATEGORY} & \textit{noun} \\ \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \textit{third} \\ \text{NUMBER} & \textit{singular} \\ \text{GENDER} & \textit{neuter} \end{bmatrix} \\ \text{CASE} & \textit{nominative} \text{ OR } \textit{accusative} \end{bmatrix}$$

At first sight Example 6.3 seems to represent a feature structure. However, the fact that the feature name [case] has two possible values (either nominative or accusative) implies that it represents two feature structures, or to put it differently, a set of feature structures. The two feature structures are identical, except for their case values: the first has a nominative case, the second an accusative case.

The reason for using disjunctive feature values is conciseness. By applying one or more disjunctive feature values per lexical item, the number of items in a lexicon can be reduced considerably.

6.2 The formalization

The formalization of Zwart's framework as described in this section formalizes Zwart's ideas on postlexicalism:

- the prelexicon will contain intrinsic as well as variable formal features
- the prelexicon will contain only semantic and formal features (no phonological features)
- a postlexicon containing phonological features 'simulates' morphological processes
- semantic and categorial features form a complex that is referred to as LC-features

Phonological, semantic and formal features In the formalization, phonological features are represented by the feature `Word` and semantic features are represented by the feature `Sememe`. `Word` takes a fully inflected

word form as its value, while `Sememe` takes a stem as its value (see also Chapter 5). The reason for this simplified treatment of phonological and semantic features is that the exact character of these features is not worked out in much detail in the minimalist literature. Furthermore, the chosen approach is effective in the formalization.

The formal features in the formalization are: `Agreement`, `Subject`, `Object`, `Person`, `Number`, `Gender`, `Case`, `Tense`, `Inversion`, `WhWord` and `Determiner`.

Features that do not belong to any of the above three types of features are: `Category`, `CompCat` and `SpecCat`. In Zwart's framework the `Category` feature is grouped together with the semantic features. This type of feature is called LC-feature (lexical categorial) by Zwart.

Postlexicalism In the formalization Zwart's postlexicalism is implemented by defining two different lexicons. The first lexicon, which according to Zwart is consulted when a lexical item enters the derivation, is called the *prelexicon*. The second lexicon, which according to Zwart is consulted after the derivation (at PF), is called the *postlexicon*.

Disjunctive feature values Before I turn to the treatment of disjunctive feature values, I will deal with the way feature structures are built up in the formalization. The function `Add` is the central function with respect to feature structures.

The function `Add`, which adds feature-value pairs to feature structures, is an indexed function. The index is given between square brackets. Each index stands for a different kind of `Add`. For instance in Definition 6.1, which contains the lexical item for the Dutch word for *book*, `Add[Sememe]` adds a value of the type `Sememe` to a feature structure yielding a feature structure, `Add[Agreement]` adds a value of the type `Agreement` to a feature structure yielding a feature structure etc.

The function `Add` is specified in such a way that adding the same feature name twice to a feature structure makes the whole lexicon undefined. A feature-value pair can only be added to a feature structure if the relevant feature name has an undefined value until then.

The feature structures in the formalization are built by starting with an empty feature structure and adding feature-value pairs to it. In Definition 6.1 we start with an empty feature structure of the category noun (N). This is relevant information because feature structures belonging to nouns do contain other features than, for example, feature structures belonging to verbs. As we saw in Chapter 5 the formalization indicates which lexical items may contain which feature-value pairs.

Also the permitted feature-value pairs for the complex feature values

of the feature names `Agreement`, `Object` and `Subject` are indicated in the formalization. For instance, the agreement value is a feature structure that is built in a way that is comparable with building a main feature structure (representing the entire noun): feature-value pairs are added to an empty feature structure. The only difference with the main feature structure is the name of the empty feature structure. Main feature structures are always introduced by `EmptyCatStruct[N]`, `EmptyCatStruct[V]`, `EmptyCatStruct[Agro]` etc., called after the category of the lexical item. Complex feature values are introduced by `EmptyStruct`. The feature name (such as the feature name `Cat` for `Category` above) need not be included because `Agreement` is already mentioned outside of the feature structure. The features in the feature structure that can be referred to as the agreement features are all permitted because `Person`, `Number` and `Gender` are defined as agreement features in the feature module (see Chapter 5).

The shared properties of different types of feature structures indicate the redundancy of the lexicon. However, the formalization does not contain inheritance principles as in feature-based theories (see [PS94], [PS87], [FPW85],[FN92]) since Zwart assumes that the lexicon contains both variable and intrinsic features. Chomsky's idea that the lexicon only contains unpredictable material resembles the idea of inheritance principles. By inheritance principles the amount of idiosyncratic information that needs to be stipulated in individual lexical items is reduced by grouping lexical items into different types. For instance, all nouns share a considerable amount of information. Of course they share the same category (N) and often (in the case of artifacts but not always in the case of personal pronouns) they have third person. Furthermore, a considerable amount of other features are shared, which may have different values for different nouns.

Now we know how feature structures are built up, we can turn to the treatment of disjunctive feature values in the formalization. There are two different ways to express sets of values in the formalization: one is a disjunction of values represented by `++` (cf. 'OR' in predicate logic), the other is in principle also a disjunction represented by `Any[Name]` which indicates the set of possible values for the feature name `Name`.

The AFSL-function `++` represents set theoretical joining. In the lexicons in our formalization the function `++` both joins the different lexical items (which themselves may be sets, as we saw above) and feature values (see Definition 6.1).

In the line `++ ( EmptyCatStruct[N]` the function `++` joins the lexical item for *book* with all the preceding lexical items, which are not represented here for the sake of simplicity. This is the same concept of the lexicon used in feature-based grammars: the lexicon is a disjunction of all its items [PS87, Page 44].

In lines such as `Add[Number] (Singular ++ Plural)` and `Add[Case] (Nominative ++ Accusative)` the function `++` joins two feature values.

Definition 6.1

```
            .
            .
            .
         ++ (  EmptyCatStruct[N]
            Add[Sememe]       "boek"
            Add[Agreement]  (  EmptyStruct
                            Add[Person]    Third
                            Add[Number]    (Singular ++
Plural)
                            Add[Gender]    Neuter
                            )
            Add[Case]       (Nominative ++ Accusative)
            Add[Determiner] (Yes ++ No)
            Add[WhWord]     No
            )
            .
            .
            .
```

A lexical item with only singleton sets as its feature values is a description of a set of feature structures with only one element, which can also be called a feature structure. All feature values that do not contain the function `++` are sets as well: they are singleton sets.

The line `Add[Number] (Singular ++ Plural)` and the line `Add[Determiner] (Yes ++ No)` in Definition 6.1 can be replaced by `Any[Number]` and `Any[Determiner]` respectively, as is shown in Definition 6.2. `Any[Name]` is applied for the disjunction of all the possible values of a given feature name. Both the feature name `Number` and the feature name `Determiner` have only two possible values, `Singular` and `Plural`, and `Yes` and `No` respectively. Since for both the feature name `Number` and the feature name `Determiner` all possible feature values are applied, Definition 6.1 can be replaced by Definition 6.2.

Definition 6.2

```
            .
            .
            .
         ++ (  EmptyCatStruct[N]
            Add[Sememe]       "boek"
            Add[Agreement]  (  EmptyStruct
                            Add[Person]    Third
```

```
                Add[Number]    Any[Number]
                Add[Gender]    Neuter
                )
    Add[Case]       (Nominative ++ Accusative)
    Add[Determiner] Any[Determiner]
    Add[WhWord]     No
    )
    .
    .
    .
```

`Any[Name]` is needed because of the nature of checking. In standard unification-based grammar the fact that a certain feature is not represented means that it can take **any** possible value, because unification succeeds when just one of the two feature structures contains a certain feature. However, checking is only possible if a certain feature is present in a given functional head as well as in the lexical constituent that checks its features against it (i.e. the lexical constituent adjoined to the functional head and/or the lexical constituent in the specifier position of the functional head) (see Chapter 5). Hence, the approach of indicating that any possible value for a given feature name can be chosen by not representing the feature-value pair at all is fatal for a theory where checking plays a role: if a feature-value pair is not visibly represented on a lexical head, the feature cannot be checked against a functional head. In Zwart's framework all functional heads need to have checked all their features against those of a lexical constituent by LF. If lexical heads might miss certain features to indicate that the feature can take any possible value, there will be a functional head that cannot check all its features, and hence LF will never be reached.

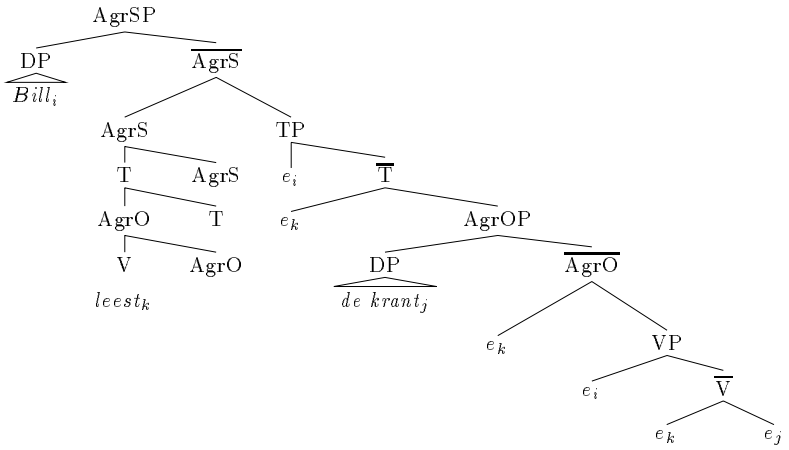
For instance, if in the sentence in Example 6.4 (Bill reads the newspaper) *krant* (newspaper) is represented by the feature structure in Definition 6.3 then the tree in Example 6.4 is not an LF-tree. AgrO cannot check its agreement number feature since the lexical item for *krant* does not contain this feature.

Definition 6.3

```
    .
    .
    .
++ (  EmptyCatStruct[N]
   Add[Sememe]       "krant"
   Add[Agreement]  (  EmptyStruct
                   Add[Person]    Third
                   Add[Gender]    Neuter
                   )
```

```
Add[Case]       (Nominative ++ Accusative)
)
.
.
.
```

Example 6.4



`Any[Name]` and `++` occur only in lexical items in the lexicon; therefore the definition of checking in Definition 5.4 on Page 83 does not deal with sets of feature values. If a lexical item is integrated in a tree, a choice has to be made between the disjuncts. Both `Any[Name]` and `++` are applied to avoid requiring that a new lexical item has to be written down for each of the disjuncts. Hence, lexical items are not always feature structures: if they contain disjunctive values (i.e. sets of values) they represent sets of features structures. We assure that a derivation tree can never contain a set of feature structures by requiring that nodes contain labels of the type `FeatureStructS` (feature structure) and not labels of the type `SetS[FeatureStructS]` (sets of feature structures).

`Any[Name]` resembles the atomic value ANY described in [Shi86, Page 43] and introduced by Kay [Kay85]. ANY is a kind of variable because it unifies with anything, just like `Any[Name]`. In a final functional structure ANY has to have been unified with something else. In our formalization something similar is the case. Only the lexicon may contain disjunctions; successful derivation trees may not.

prelexicon The definition of the prelexicon, including only the first lexical item for the sake of conciseness, is given in Definition 6.4.

Definition 6.4

```
MODULE PreLexiconZW

IMPORT FeaturesM

IMPORT SetM[FeatureStructS]

OBJ    PreLexicon : SetS[FeatureStructS]
       = ++ (  EmptyCatStruct[N]
            Add[Sememe]       "boek"
            Add[Agreement]  (  EmptyStruct
                               Add[Person]    Third
                               Add[Number]    Any[Number]
                               Add[Gender]    Neuter
                               )
            Add[Case]       (Nominative ++ Accusative)
            Add[Determiner] Any[Determiner]
            Add[WhWord]     No
            )
            .
            .
            .

END MODULE
```

In Definition 6.4 we see that, firstly, the module where all features and their values are introduced is imported: `IMPORT FeaturesM`. Of course, this is done because a prelexicon is a set of feature structures with feature names and feature values as its main building blocks. Next, we import all the specified information about sets by the line: `IMPORT SetM[FeatureStructS]`. Finally, it is declared that the prelexicon is an object existing of sets of feature structures: `OBJ PreLexicon :  SetS[FeatureStructS]`.

The prelexicon is connected with the rest of the formalization via the functions `XFeatures` in the module on $\overline{\text{X}}$-Theory (see Chapter 7). This axiom says that leaves with `BarLevel 0` have a feature structure that is a member of the lexicon. What has to be taken into account is that nodes of trees may never contain disjunctive feature values. In the formalization typing takes care of this: nodes of trees contain feature structures (`FeatureStructS`), not sets of feature structures (`SetS[FeatureStructS]`). As we said earlier in this section, we must distinguish between a description of a set of feature structures (which contains disjunctive feature values) and a single feature value (which may not contain disjunctive feature values). Nodes of trees may not contain sets of feature structures, as they are not of the right type.

The fact that sets of feature structures are not appropriate in nodes of trees is a consequence of the necessity of feature checking. Checking cannot take place if, for instance, it is not clear if a lexical item has nominative or accusative case. When a feature is checked it must have a singleton feature value.

By the application of the prelexicon in the formalization, we simulate Zwart's fixation of variable features when the lexical element enters the derivation. Namely, elements from the prelexicon do not contain multiple values once they are applied in an LF-tree.

Postlexicon Since we apply postlexicalism in the formalization we need two different lexicons. The first lexicon (prelexicon) contains one lexical entry per stem (sememe). We will see that the lexicon that is consulted at PF (postlexicon) contains one item per inflected form, for instance one for *zij* (she) and one for *haar* (her). The prelexicon does not contain `Word` features as opposed to the postlexicon. This is because Zwart argues that the prelexicon is a more or less language independent lexicon, with only formal (intrinsic as well as variable) and semantic features. In our formalization the `Sememe` feature represents the semantic features, as the literature is not specific enough about the nature of semantic features. The application of the `Sememe` feature instead of semantic features causes the prelexicon to be less language independent than it would be if actual semantic features were used, as the `Sememe` feature takes a language specific stem as its value. The `Word` feature in the postlexicon replaces phonological features. The Minimalist Program is not explicit about phonological features. Therefore, we decided to apply the `Word` feature instead of phonological features, as the word is a representation of how to pronounce a lexical item.

The postlexicon is the more extensive of the two lexicons, because it often contains more than one form of the same paradigm. For instance, the stem *zij* (she) appears in two different forms: once as *zij* and once as *haar* (see Definition 6.5).

Definition 6.5

```
MODULE PostLexiconZW

IMPORT FeaturesM

IMPORT SetM[FeatureStructS]

OBJ PostLexicon : SetS[FeatureStructS]

AXIOM  PostLexicon
       =    .
```

```
            .
            .
         ++ (  EmptyCatStruct[N]
            Add[Stem]       "zij"
            Add[Word]       "zij"
            Add[Agreement]  (  EmptyStruct
                            Add[Person]    Third
                            Add[Number]    Singular
                            Add[Gender]    Feminine
                            )
            Add[Case]       Nominative
            Add[Determiner] No
            Add[WhWord]     No
            )
         ++ (  EmptyCatStruct[N]
            Add[Stem]       "zij"
            Add[Word]       "haar"
            Add[Agreement]  (  EmptyStruct
                            Add[Person]    Third
                            Add[Number]    Singular
                            Add[Gender]    Feminine
                            )
            Add[Case]       Accusative
            Add[Determiner] No
            Add[WhWord]     No
            )
            .
            .
            .
END MODULE
```

The part above the axiom in Definition 6.5 is comparable with the beginning of the prelexicon module.

The postlexicon is connected with the rest of the formalization at PF by the function `LookUpWord` (see Chapter 9): when a tree is spelled out, the postlexicon is consulted. In a tree that is spelled out there are no phonetic/word features to be found, as the prelexicon, on which the LF-tree is based, does not contain `Word` features. However, the prelexicon does contain `Sememe` features. At PF, entries that can be unified with the features of the heads of the tree are selected from the postlexicon. A sentence that is spelled out is a list of words that is based on the `Word` values of those elements. Sometimes more than one entry per head is selected from the postlexicon. In such a case the most specific entry is chosen. The most specific entry is the entry with most features that can still be checked against the features of the head from the tree that is spelled out. For example, the paradigm *kus* (kiss) has two occurrences that can be checked against a feature structure with a first person singular form. The first has the word feature *kus* and contains an explicit person feature with the value *first*. The second has the word feature *kust* and does not contain a person

feature, which means that it unifies with any person value (cf. Examples 6.1 and 6.2). The most specific occurrence of the two is, of course, the first. Namely, the second does not have a person feature at all. Therefore the first occurrence of the `Sememe` *kus* with the word value *kus* is spelled out. Hence, if more than one matching feature structure is found in the postlexicon, the most specific feature structure is selected by the function `LookUpWord`.

In the postlexicon, the Any-value is not applied when a feature can occur with any possible value. Not listing the feature at all is more economical according to Zwart. Moreover, it would be incorrect to use the Any-value for this objective. If we would apply Any-values in the case of the above example, the postlexicon would contain three occurrences of the word *kust*, of which one is first person. This is a form that does not occur in Dutch: *Ik kust hem* (I kisses him) is not a grammatical Dutch sentence. If this form did exist it would be impossible to select the most specific form from the postlexicon because *kus* and *kust* are equally specific: they both contain a person feature with the value *first*.

Therefore Zwart's idea of not giving a person feature at all for the form *kust* is preferred here. If an LF-tree includes a head with a feature structure containing a stem *kus* and a person value *first*, the form that is spelled out is *kus*. The reason for this is that the postlexicon contains two matching feature structures, one with the word value *kus* and the other with the word value *kust*, of which the former is the more specific because it contains a person feature with the value *first*, while the latter has no person feature.

The idea of selecting the most specific form is well-known in linguistic literature. Aronoff [Aro76] describes the principle of Morphological Blocking, which implies that the existence of a more highly specified form in the lexicon excludes the selection of a less highly specified form. Morphological Blocking is considered to be a consequence of the Elsewhere Condition. The Elsewhere Condition requires that rules with more specific constraints apply before rules with more general constraints (cf. [And69, Kip73, And86]).

What is problematic is that it is not always possible to indicate which feature structure is the more specific of two. For instance, two feature structures that both consist of three feature-value pairs might share two feature-value pairs while the third is different for both. Such cases do not occur in the fragment that our formalization describes. However, theoretically it might happen that it is impossible to indicate which of two feature structures is the most specific.

Note that the use of the postlexicon more or less denies the role of inflectional morphology. Inflectional morphology enables us to derive inflected forms from a stem on the basis of inflectional features such as [number], [person] and [tense]. In the approach chosen here the paradigm belonging to a stem consists of a group of postlexical items in the lexicon from which

we can make a choice.

6.3 The lexicon in Chomsky's 1993 framework

In the formalization of Chomsky's 1993 framework there only is one lexicon. As we saw earlier this lexicon contains semantic, formal and phonological features. As in the formalization of Zwart's framework the lexicon is connected with the rest of the formalization via $\overline{\text{X}}$-Theory. The `Word` features in the lexicon represent the phonological features of the lexical items. At PF these are the features that are spelled out.

6.4 Summary

In this chapter I discussed the treatment of the lexicon in the formalization. There are differences between the treatment of the lexicon in Zwart's framework and in Chomsky's 1993 framework. These differences are caused by the *postlexicalist* approach chosen by Zwart. In this approach, the main idea is that phonological features are first introduced after the derivation. The lexicon only contains semantic and formal features. In Chomsky's framework the lexicon contains semantic and formal as well as phonological features. In Zwart's framework an extra lexicon is introduced which is consulted at PF and which **does** contain phonological features. The selection from the extra lexicon (which is called postlexicon in the formalization) is based on the formal and semantic features from the lexical heads in the LF-tree. The idea behind Zwart's postlexicalism is that it is more economical to introduce the phonological features, which are not needed in the course of the derivation, after the derivation.